autogluon_each_location

October 26, 2023

1 Config

[1]:
```python
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*10
presets = "best_quality"#'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

drop_night_outliers = True
drop_null_outliers = False

to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
 ↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
 ↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
 ↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
 ↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
 ↪"pressure_100m:hPa"]

#to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:
 ↪ms",␣"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:
 ↪cm",␣"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:
 ↪mm",␣"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm",␣
 ↪"absolute_humidity_2m:gm3", "air_density_2m:kgm3"]

use_groups = False
n_groups = 8

# auto_stack = True
num_stack_levels = 0
num_bag_folds = None# 8
```

```
num_bag_sets = None#20

use_tune_data = True
use_test_data = True
#tune_and_test_length = 0.5 # 3 months from end
# holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[2]:
```
import pandas as pd
import numpy as np



import warnings
warnings.filterwarnings("ignore")


def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()
```

```
    # Create a set for constant-time lookup
    index_set = set(X.index)

    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
↪loc[shifted_indices[shifted_indices.isin(index_set)]][columns]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
↪columns]

    date_calc = None
    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']




    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
```

```
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
```

```
        X_test["estimated_diff_hours"] = (X_test.index - pd.
↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1


    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")

    print(f"Size of estimated after dropping nans:␣
↪{len(X_train[X_train['is_estimated']==1].dropna(subset=['y']))}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
```

```python
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Size of estimated after dropping nans: 4418
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
```

```
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Size of estimated after dropping nans: 3625
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
Size of estimated after dropping nans: 2954
```

2.1 Feature enginering

2.1.1 Remove anomalies

[3]:

```
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
 ↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}
```

```
        for idx in x.index:
            value = x[idx]
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
 ↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

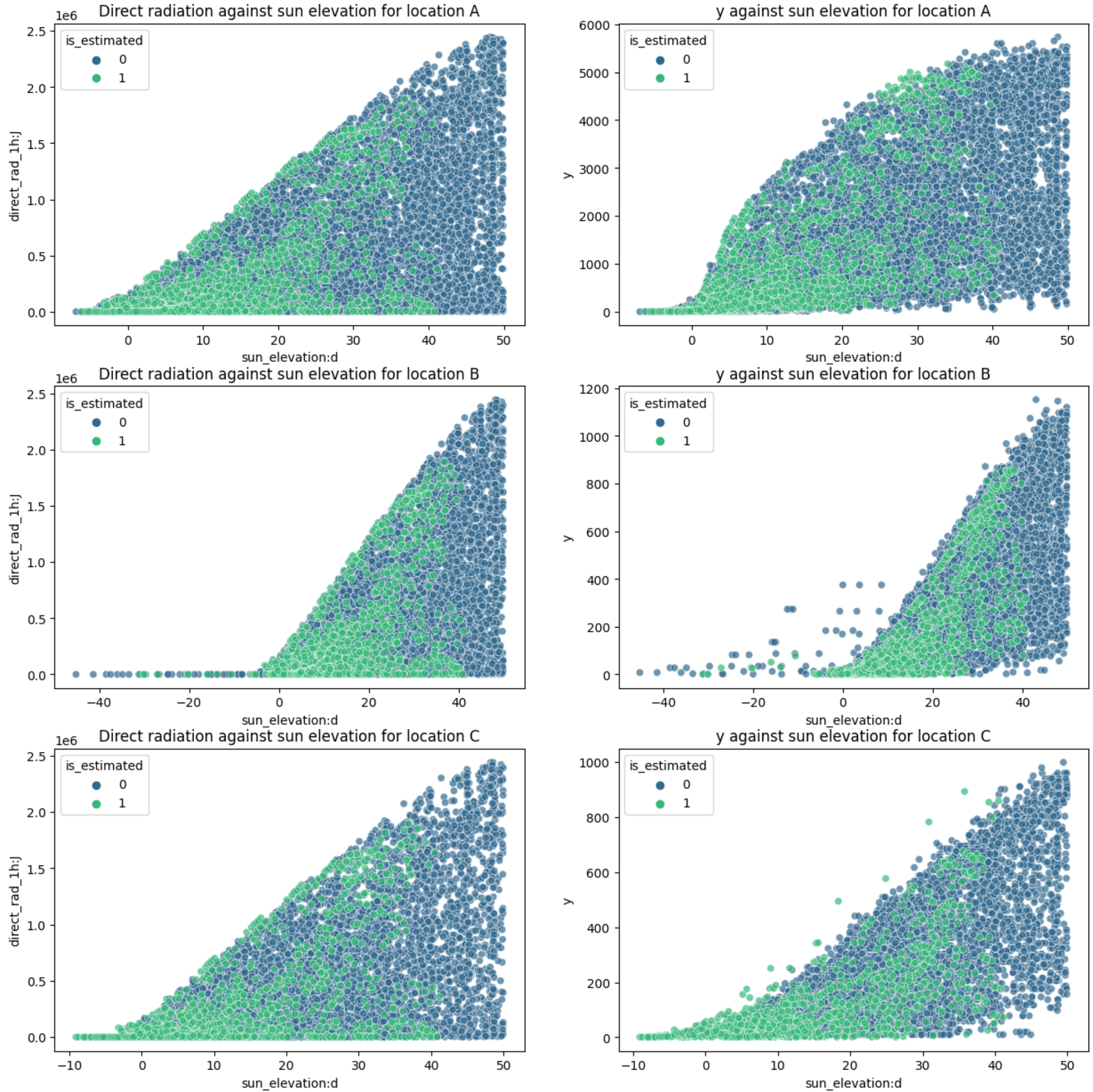
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[4]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
 ↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

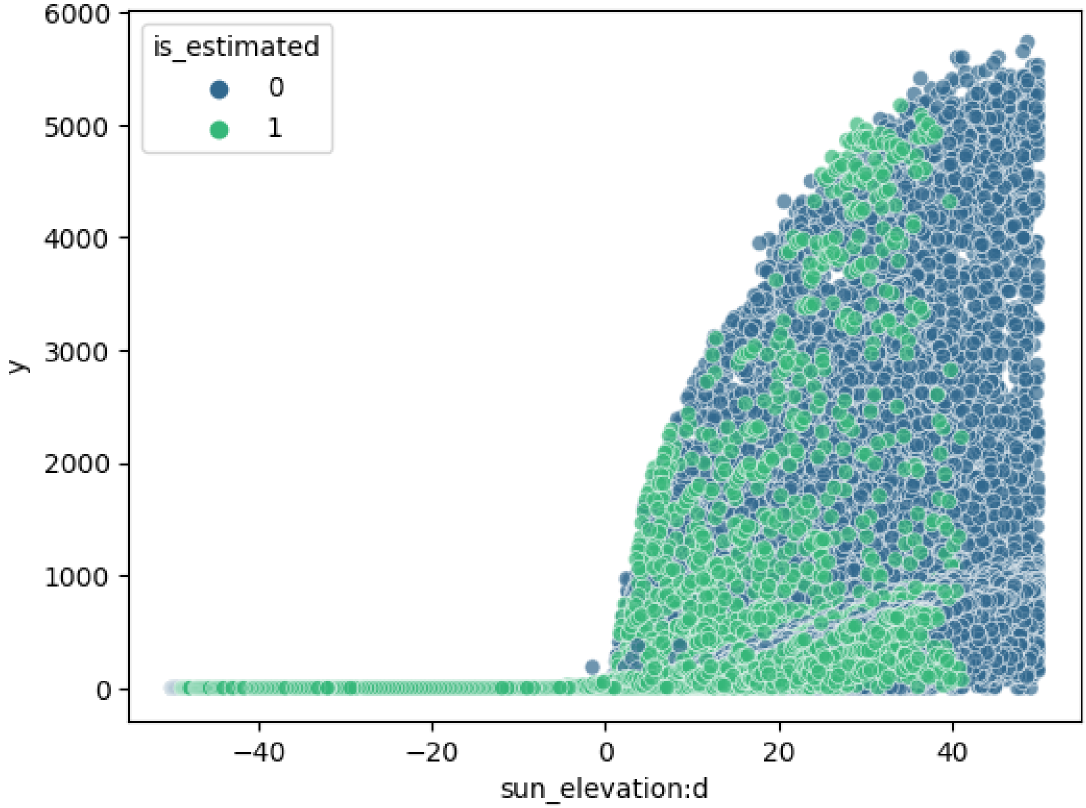
```
Dropped rows:  9293
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated",␣
 ↪palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for␣
 ↪location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location],␣
 ↪x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```

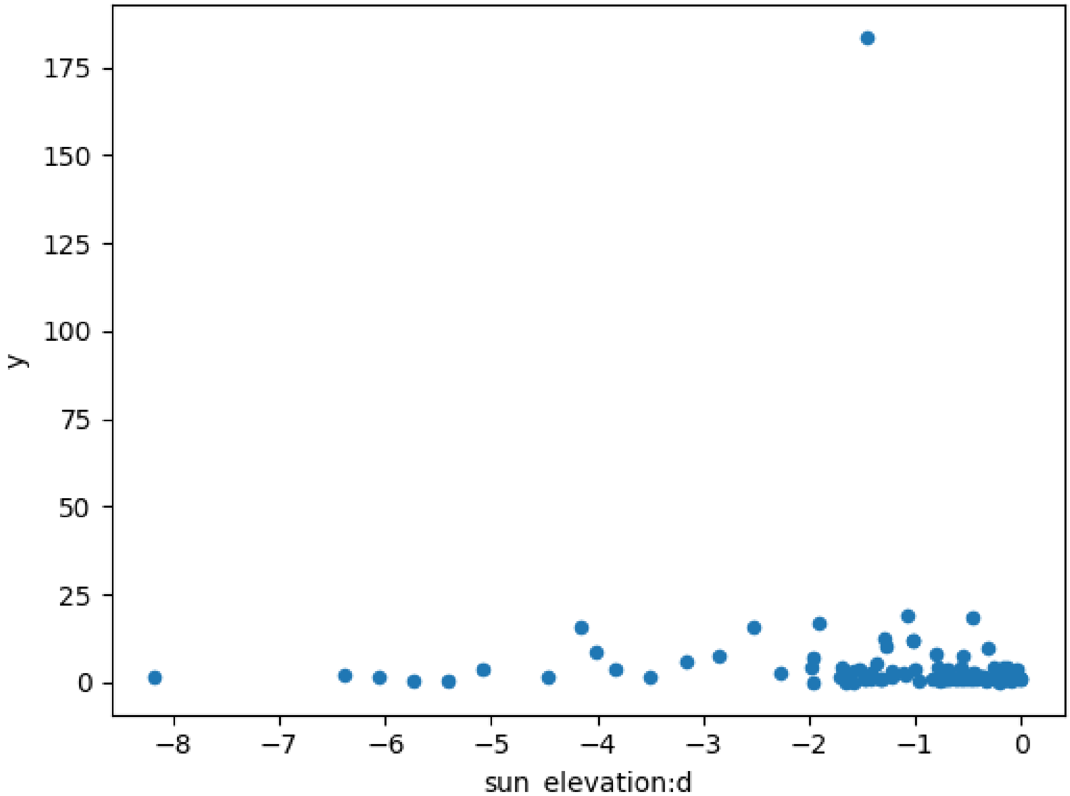


[6]:
```
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=
  ↪thresh) & (X_train["y"] >= 0.1)
if drop_night_outliers:
    X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",
  ↪palette="viridis", alpha=0.7)
plt.show()
```



[7]:
```
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

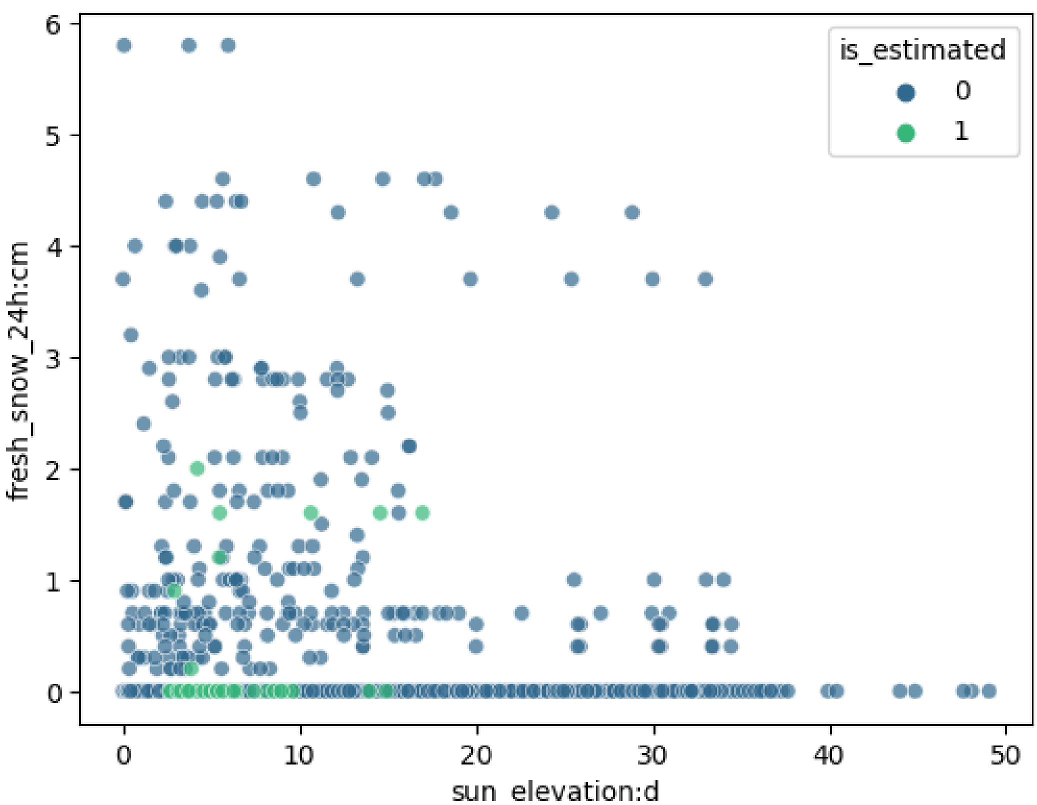
[7]:
```
<AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>
```



[8]:
```
# set y to nan where y is 0, but direct_rad_1h:J or diffuse_rad_1h:J are > 0␣
↪(or some threshold)
threshold_direct = X_train["direct_rad_1h:J"].max() * 0.01
threshold_diffuse = X_train["diffuse_rad_1h:J"].max() * 0.01
print(f"Threshold direct: {threshold_direct}")
print(f"Threshold diffuse: {threshold_diffuse}")

mask = (X_train["y"] == 0) & ((X_train["direct_rad_1h:J"] > threshold_direct) |␣
↪(X_train["diffuse_rad_1h:J"] > threshold_diffuse)) & (X_train["sun_elevation:
↪d"] > 0) & (X_train["fresh_snow_24h:cm"] < 6) & (X_train[['fresh_snow_12h:
↪cm', 'fresh_snow_1h:cm',  'fresh_snow_3h:cm', 'fresh_snow_6h:cm']].
↪sum(axis=1) == 0)
print(len(X_train[mask]))

#print(X_train[mask][[x for x in X_train.columns if "snow" in x]])

# show plot where mask is true
#sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="y",␣
↪hue="is_estimated", palette="viridis", alpha=0.7)
```

```
sns.scatterplot(data=X_train[mask], x="sun_elevation:d", y="fresh_snow_24h:cm",␣
 ↪hue="is_estimated", palette="viridis", alpha=0.7)
plt.show()

#sns.scatterplot(data=X_train[mask], x="fresh_snow_24h:cm",␣
 ↪y="total_cloud_cover:p", hue="is_estimated", palette="viridis", alpha=0.7)



# set y to nan where mask
if drop_null_outliers:
    X_train.loc[mask, "y"] = np.nan

# show how many rows for each location, and for estimated and not estimated
X_train[mask].groupby(["location", "is_estimated"]).count()["direct_rad_1h:J"]
```

```
Threshold direct: 24458.97
Threshold diffuse: 11822.505000000001
2599
```



[8]:
```
location  is_estimated
A         0                 87
          1                 10
B         0               1250
          1                 32
C         0               1174
          1                 46
Name: direct_rad_1h:J, dtype: int64
```

[9]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  1876
```

2.1.2 Other stuff

[10]:
```
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())


# If the "sample_weight" column is present and weight_evaluation is True,␣
↪multiply sample_weight with sample_weight_may_july if the ds is between␣
↪05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to␣
↪X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
↪((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *=␣
↪sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) |␣
↪((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))
```

```
if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name␣
↪of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups),␣
↪repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())






X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.625
air_density_2m:kgm3                  1.2215
ceiling_height_agl:m            3644.050049
clear_sky_energy_1h:J            2896336.75
clear_sky_rad:W                  753.849976
cloud_base_agl:m                3644.050049
dew_or_rime:idx                         0.0
```

```
dew_point_2m:K                       280.475006
diffuse_rad:W                        127.475006
diffuse_rad_1h:J                     526032.625
direct_rad:W                              488.0
direct_rad_1h:J                     1718048.625
effective_cloud_cover:p               18.200001
elevation:m                                 6.0
fresh_snow_12h:cm                           0.0
fresh_snow_1h:cm                            0.0
fresh_snow_24h:cm                           0.0
fresh_snow_3h:cm                            0.0
fresh_snow_6h:cm                            0.0
is_day:idx                                  1.0
is_in_shadow:idx                            0.0
msl_pressure:hPa                    1026.775024
precip_5min:mm                              0.0
precip_type_5min:idx                        0.0
pressure_100m:hPa                   1013.599976
pressure_50m:hPa                    1019.599976
prob_rime:p                                 0.0
rain_water:kgm2                             0.0
relative_humidity_1000hPa:p           53.825001
sfc_pressure:hPa                    1025.699951
snow_density:kgm3                           NaN
snow_depth:cm                               0.0
snow_drift:idx                              0.0
snow_melt_10min:mm                          0.0
snow_water:kgm2                             0.0
sun_azimuth:d                        222.089005
sun_elevation:d                       44.503498
super_cooled_liquid_water:kgm2              0.0
t_1000hPa:K                          286.700012
total_cloud_cover:p                   18.200001
visibility:m                           52329.25
wind_speed_10m:ms                           2.6
wind_speed_u_10m:ms                        -1.9
wind_speed_v_10m:ms                       -1.75
wind_speed_w_1000hPa:ms                     0.0
is_estimated                                  0
y                                       4367.44
location                                      A
Name: 2019-06-11 13:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 23:00:00                       7.7               1.2235

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
```

```
2019-06-02 23:00:00           1689.824951                   0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 23:00:00              0.0       1689.824951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                    …
2019-06-02 23:00:00      280.299988            0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 23:00:00   286.899994                100.0  33770.648438

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 23:00:00               3.35                -3.35

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 23:00:00                0.275                      0.0

                     is_estimated    y  location
ds
2019-06-02 23:00:00             0  0.0         A

[1 rows x 48 columns]
```

[11]:

```
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
    plt.show()
```

[12]:

```
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd
```

```python
def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
 ↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
 ↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
 ↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
```

```
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
↪]])

    return output_data1, output_data2
```

[13]:
```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
estimated_set = TabularDataset(data[data["ds"] >= split_time]) # only estimated

test_set = pd.DataFrame()
tune_set = pd.DataFrame()
new_train_set = pd.DataFrame()

if not use_tune_data:
    raise Exception("Not implemented")

for location in locations:
    loc_data = data[data["location"] == location]
    num_train_rows = len(loc_data)

    tune_rows = 1500.0 # 2500.0
    if use_test_data:
        tune_rows = 1880.0#max(3000.0,␣
↪len(estimated_set[estimated_set["location"] == location]))
```

```
    holdout_frac = max(0.01, min(0.1, tune_rows / num_train_rows)) *␣
↪num_train_rows / len(estimated_set[estimated_set["location"] == location])

    print(f"Size of estimated for location {location}:␣
↪{len(estimated_set[estimated_set['location'] == location])}. Holdout frac␣
↪should be % of estimated: {holdout_frac}")

    # shuffle and split data
    loc_tune_set, loc_new_train_set =␣
↪split_and_shuffle_data(estimated_set[estimated_set['location'] == location],␣
↪40, holdout_frac)
    print(f"Length of location tune set : {len(loc_tune_set)}")
    new_train_set = pd.concat([new_train_set, loc_new_train_set])

    if use_test_data:
        loc_test_set, loc_tune_set = split_and_shuffle_data(loc_tune_set, 40, 0.
↪2)
        test_set = pd.concat([test_set, loc_test_set])

    tune_set = pd.concat([tune_set, loc_tune_set])




print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = tune_set

# number of rows in tuning data for each location
print("Shapes of tuning data", tuning_data.groupby('location').size())



if use_test_data:
    test_data = test_set
    print("Shape of test", test_data.shape[0])
```

```
train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of
↪rows in the training (or tuning) data.
if weight_evaluation:
    # ensure sample weights for data sum to the number of rows in the tuning /
↪train data.
    tuning_data = normalize_sample_weights_per_location(tuning_data)
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
tuning_data = TabularDataset(tuning_data)

if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Size of estimated for location A: 4214. Holdout frac should be % of estimated:
0.4461319411485524
Length of location tune set : 1846
Size of estimated for location B: 3533. Holdout frac should be % of estimated:
0.5321256722332296
Length of location tune set : 1846
Size of estimated for location C: 2923. Holdout frac should be % of estimated:
0.6431748203900103
Length of location tune set : 1841
Length of train set before adding test set 77247
Length of train set after adding test set 82384
Shapes of tuning data location
A    1485
B    1485
C    1481
dtype: int64
Shape of test 1082
```

3 Quick EDA

[14]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,
↪label="y", sample=None)
```

[15]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[16]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 116
Now creating submission number: 117
New filename: submission_117
```

[17]:
```
predictors = [None, None, None]
```

[18]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
```

```python
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
 ↪drop(columns=["ds"]),
        time_limit=time_limit,
        presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
 ↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
 ↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
 ↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_117_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   189.14 GB / 315.93 GB (59.9%)
Train Data Rows:    30934
Train Data Columns: 32
```

```
Tuning Data Rows:    1485
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 673.41535, 1195.24)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    132333.58 MB
        Train Data (Original)  Memory Usage: 9.92 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…

Training model for location A…

        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
```

```
data.
	Train Data (Processed) Memory Usage: 7.55 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.15s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
	This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
	To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
	'NN_TORCH': {},
	'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
	'CAT': {},
	'XGB': {},
	'FASTAI': {},
	'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
	'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
	'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.85s of the
599.85s of remaining time.
	-191.231	 = Validation score   (-mean_absolute_error)
	0.03s	 = Training   runtime
	0.37s	 = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.37s of the
599.37s of remaining time.
	-192.918	 = Validation score   (-mean_absolute_error)
	0.03s	 = Training   runtime
	0.37s	 = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 598.91s of the
598.9s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-89.2737	 = Validation score   (-mean_absolute_error)
	28.13s	 = Training   runtime
```

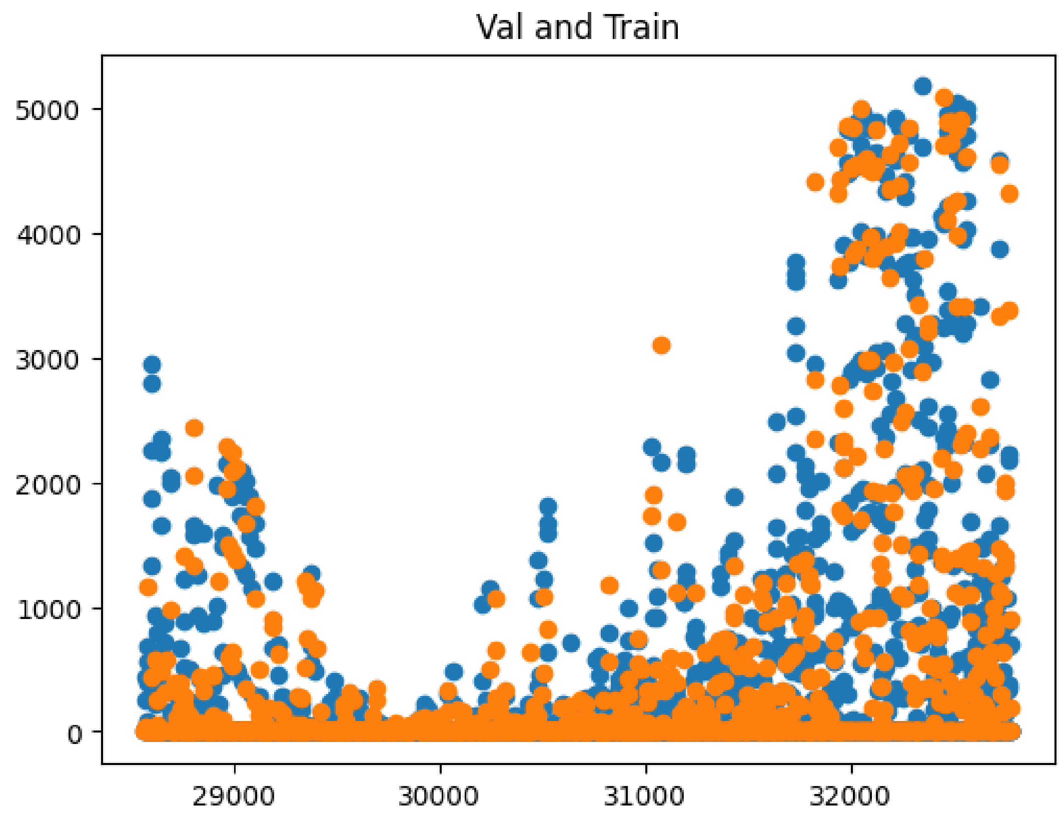
```
        14.39s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 561.64s of the
561.64s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -92.2389         = Validation score   (-mean_absolute_error)
        21.63s   = Training   runtime
        5.57s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 536.36s of
the 536.36s of remaining time.
        -99.7294         = Validation score   (-mean_absolute_error)
        7.24s    = Training   runtime
        1.06s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 526.91s of the
526.91s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -99.4662         = Validation score   (-mean_absolute_error)
        187.92s  = Training   runtime
        0.08s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 337.82s of the
337.82s of remaining time.
        -102.708         = Validation score   (-mean_absolute_error)
        1.51s    = Training   runtime
        1.1s     = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 334.04s of
the 334.04s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -101.9672        = Validation score   (-mean_absolute_error)
        38.63s   = Training   runtime
        0.46s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 292.68s of the
292.68s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -98.2846         = Validation score   (-mean_absolute_error)
        45.02s   = Training   runtime
        1.55s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 243.96s of the
243.96s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -88.706  = Validation score   (-mean_absolute_error)
        111.92s  = Training   runtime
        0.31s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 130.66s of the
130.66s of remaining time.
```

```
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -90.3269         = Validation score   (-mean_absolute_error)
        93.36s   = Training   runtime
        15.91s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
28.52s of remaining time.
        -84.4359         = Validation score   (-mean_absolute_error)
        0.43s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 571.93s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_117_A/")
Evaluation: mean_absolute_error on test data: -106.6746501360557
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -106.6746501360557,
    "root_mean_squared_error": -342.66027511438546,
    "mean_squared_error": -117416.06414146634,
    "r2": 0.8157253903077522,
    "pearsonr": 0.9084585857616528,
    "median_absolute_error": -2.065485715866089
}

Evaluation on test data:
-106.6746501360557
```

[19]:

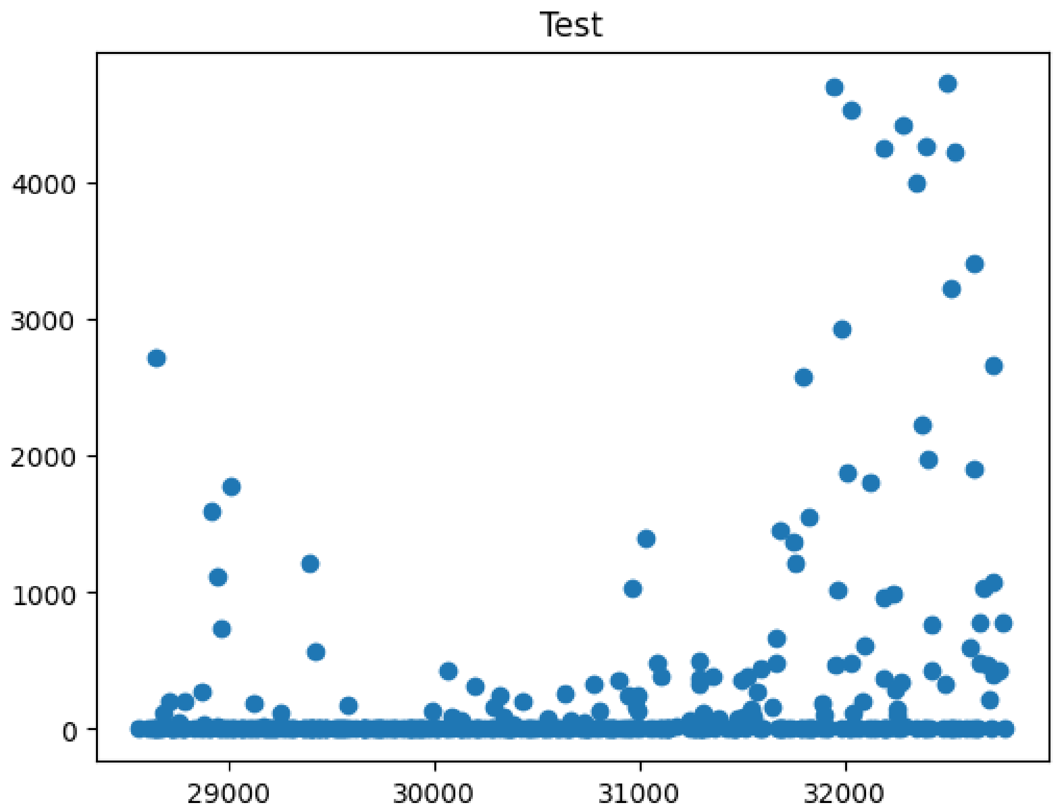
```
import matplotlib.pyplot as plt
leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_tune_data:
        plt.scatter(train_data[(train_data["location"] == loc) &
 ↪(train_data["is_estimated"]==True)]["y"].index,
 ↪train_data[(train_data["location"] == loc) &
 ↪(train_data["is_estimated"]==True)]["y"])
        plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.title("Val and Train")
        plt.show()

        if use_test_data:
            lb = predictors[i].leaderboard(test_data[test_data["location"] ==
 ↪loc])
            lb["location"] = loc
```

```
            plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
↪test_data[test_data["location"] == loc]["y"])
            plt.title("Test")

            return lb

    return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```




```
                     model  score_test   score_val  pred_time_test
pred_time_val    fit_time  pred_time_test_marginal  pred_time_val_marginal
fit_time_marginal  stack_level  can_infer  fit_order
0         LightGBMXT_BAG_L1 -104.707914  -89.273737        0.878482
14.389864   28.132791                 0.878482              14.389864
28.132791            1       True           3
1       WeightedEnsemble_L2 -106.674650  -84.435852        4.095314
30.609674  233.845275                 0.003637               0.000643
0.427538             2       True          12
2     NeuralNetTorch_BAG_L1 -112.835713  -88.706020        0.158723
0.305625  111.923088                 0.158723              0.305625
```

```
111.923088            1       True         10
3     LightGBMLarge_BAG_L1 -116.944205  -90.326904        3.054472
15.913542   93.361858                3.054472                15.913542
93.361858            1       True         11
4          LightGBM_BAG_L1 -118.219832  -92.238938        0.565693
5.567097   21.628919               0.565693                5.567097
21.628919            1       True         4
5   NeuralNetFastAI_BAG_L1 -118.952940 -101.967194        0.187788
0.464371   38.632999               0.187788                0.464371
38.632999            1       True         8
6          CatBoost_BAG_L1 -119.620368  -99.466161        0.058971
0.081967  187.916759               0.058971                0.081967
187.916759            1       True         6
7           XGBoost_BAG_L1 -121.866670  -98.284592        0.330251
1.549180   45.019486               0.330251                1.549180
45.019486            1       True         9
8      ExtraTreesMSE_BAG_L1 -130.014875 -102.707958        0.551604
1.104205    1.505122               0.551604                1.104205
1.505122            1       True         7
9   RandomForestMSE_BAG_L1 -131.765656  -99.729353        0.563023
1.059959    7.244803               0.563023                1.059959
7.244803            1       True         5
10   KNeighborsDist_BAG_L1 -189.567265 -192.917996        0.014678
0.366823    0.029757               0.014678                0.366823
0.029757            1       True         2
11   KNeighborsUnif_BAG_L1 -191.283846 -191.231007        0.172469
0.365815    0.030283               0.172469                0.365815
0.030283            1       True         1
```



[20]:
```
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

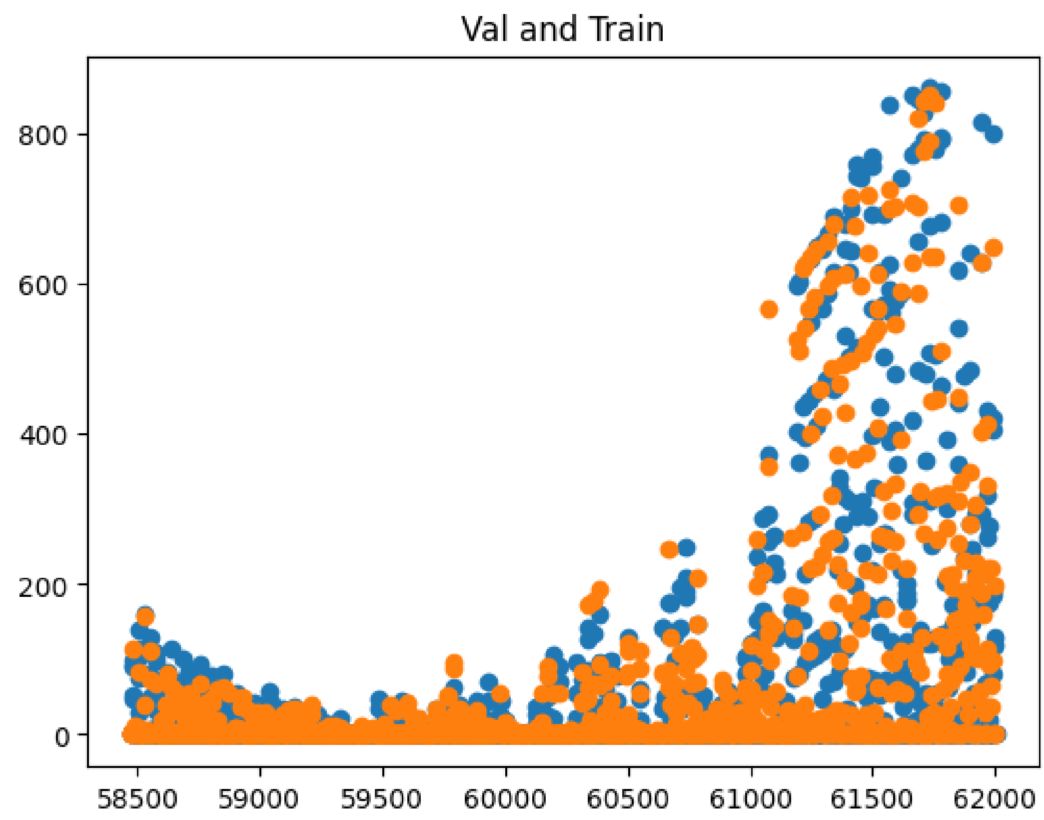
```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s

Training model for location B…

AutoGluon will save models to "AutogluonModels/submission_117_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   186.93 GB / 315.93 GB (59.2%)
Train Data Rows:    27377
Train Data Columns: 32
Tuning Data Rows:    1485
Tuning Data Columns: 32
```

```
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 98.11625, 206.48535)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    130186.16 MB
        Train Data (Original)  Memory Usage: 8.83 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 6.72 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.15s …
AutoGluon will gauge predictive performance using evaluation metric:
```

```
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.85s of the
599.85s of remaining time.
        -28.5444         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.33s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.43s of the
599.43s of remaining time.
        -28.798  = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.34s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 599.01s of the
599.01s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.0816         = Validation score   (-mean_absolute_error)
        28.31s   = Training   runtime
        14.71s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 566.26s of the
566.26s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

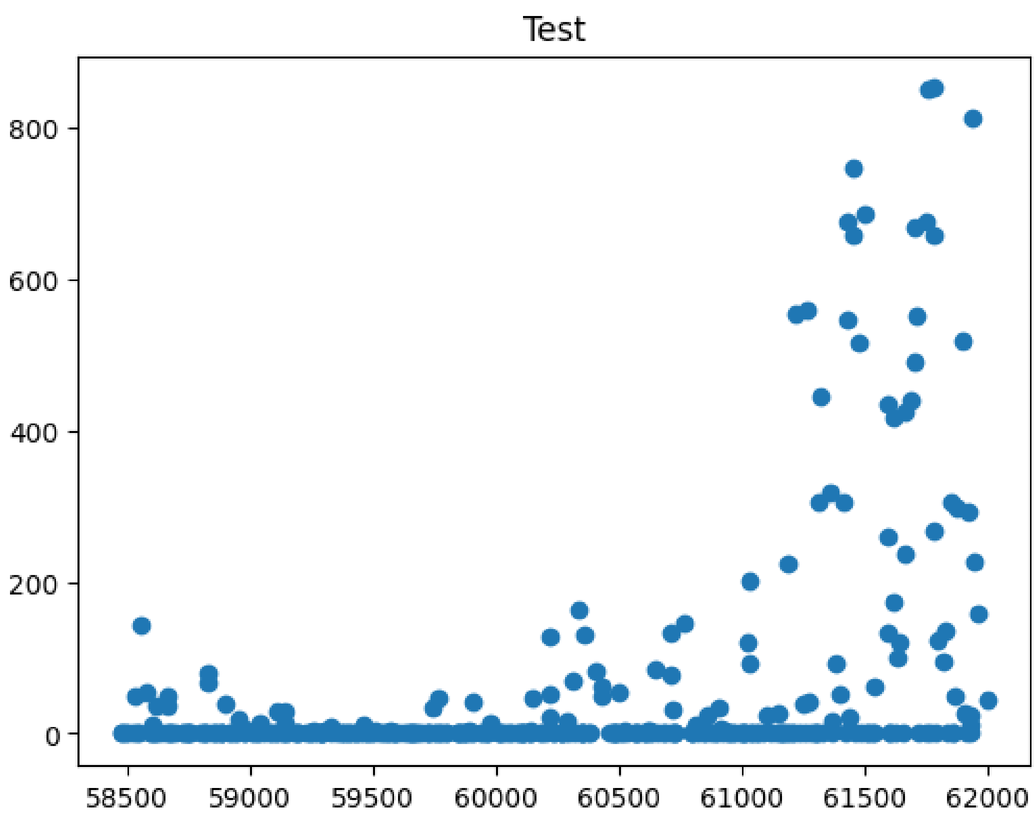
```
ParallelLocalFoldFittingStrategy
        -14.7239         = Validation score   (-mean_absolute_error)
        27.51s   = Training   runtime
        11.03s   = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 534.29s of
the 534.29s of remaining time.
        -16.0635         = Validation score   (-mean_absolute_error)
        5.66s    = Training   runtime
        0.88s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 526.91s of the
526.91s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.9574         = Validation score   (-mean_absolute_error)
        189.65s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 336.04s of the
336.04s of remaining time.
        -15.5342         = Validation score   (-mean_absolute_error)
        1.17s    = Training   runtime
        0.91s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 333.09s of
the 333.09s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.8576         = Validation score   (-mean_absolute_error)
        34.95s   = Training   runtime
        0.43s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 296.62s of the
296.62s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.6132         = Validation score   (-mean_absolute_error)
        44.16s   = Training   runtime
        3.7s     = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 249.1s of the
249.1s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.0009         = Validation score   (-mean_absolute_error)
        149.79s  = Training   runtime
        0.28s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 97.92s of the
97.91s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.2116         = Validation score   (-mean_absolute_error)
        83.09s   = Training   runtime
```

```
        27.37s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
3.24s of remaining time.
        -12.7267         = Validation score   (-mean_absolute_error)
        0.41s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 597.19s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_117_B/")
Evaluation: mean_absolute_error on test data: -11.431644027172322
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -11.431644027172322,
    "root_mean_squared_error": -32.4833161353096,
    "mean_squared_error": -1055.1658271464648,
    "r2": 0.9545740874376115,
    "pearsonr": 0.9770449143351182,
    "median_absolute_error": -0.28227362036705017
}

Evaluation on test data:
-11.431644027172322
```




```
                      model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0       WeightedEnsemble_L2  -11.431644 -12.726710        5.119251      44.594485
303.387221                 0.004865                0.000558           0.413515
2        True         12
1           LightGBM_BAG_L1  -11.777448 -14.723882        0.854106      11.032186
27.507390                 0.854106               11.032186          27.507390
1        True          4
2      LightGBMLarge_BAG_L1  -11.796480 -14.211572        3.002045      27.371333
83.085582                 3.002045               27.371333          83.085582
1        True         11
3         LightGBMXT_BAG_L1  -11.854513 -14.081634        0.984789      14.712231
28.310808                 0.984789               14.712231          28.310808
1        True          3
4           CatBoost_BAG_L1  -11.929600 -14.957434        0.063016       0.085552
189.653611                 0.063016                0.085552         189.653611
1        True          6
5     NeuralNetTorch_BAG_L1  -12.017231 -13.000913        0.155384       0.281691
149.791439                 0.155384                0.281691         149.791439
1        True         10
```

```
6            XGBoost_BAG_L1  -12.585263 -14.613203        0.740595       3.695811
44.158752                0.740595                 3.695811          44.158752
1        True           9
7    NeuralNetFastAI_BAG_L1  -12.854742 -13.857616        0.182944       0.432665
34.953278                0.182944                 0.432665          34.953278
1        True           8
8      ExtraTreesMSE_BAG_L1  -12.996028 -15.534221        0.404173       0.912390
1.174805                0.404173                 0.912390          1.174805
1        True           7
9    RandomForestMSE_BAG_L1  -13.060859 -16.063545        0.385051       0.883616
5.657793                0.385051                 0.883616          5.657793
1        True           5
10    KNeighborsDist_BAG_L1  -23.570591 -28.797985        0.012008       0.336538
0.024547                0.012008                 0.336538          0.024547
1        True           2
11    KNeighborsUnif_BAG_L1  -24.697224 -28.544405        0.012848       0.327027
0.024534                0.012848                 0.327027          0.024534
1        True           1
```



[21]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
```

```
leaderboards[2] = leaderboard_for_location(2, loc)
```

```
Presets specified: ['best_quality']
Stack configuration (auto_stack=True): num_stack_levels=0, num_bag_folds=8,
num_bag_sets=20
Beginning AutoGluon training … Time limit = 600s
AutoGluon will save models to "AutogluonModels/submission_117_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
```

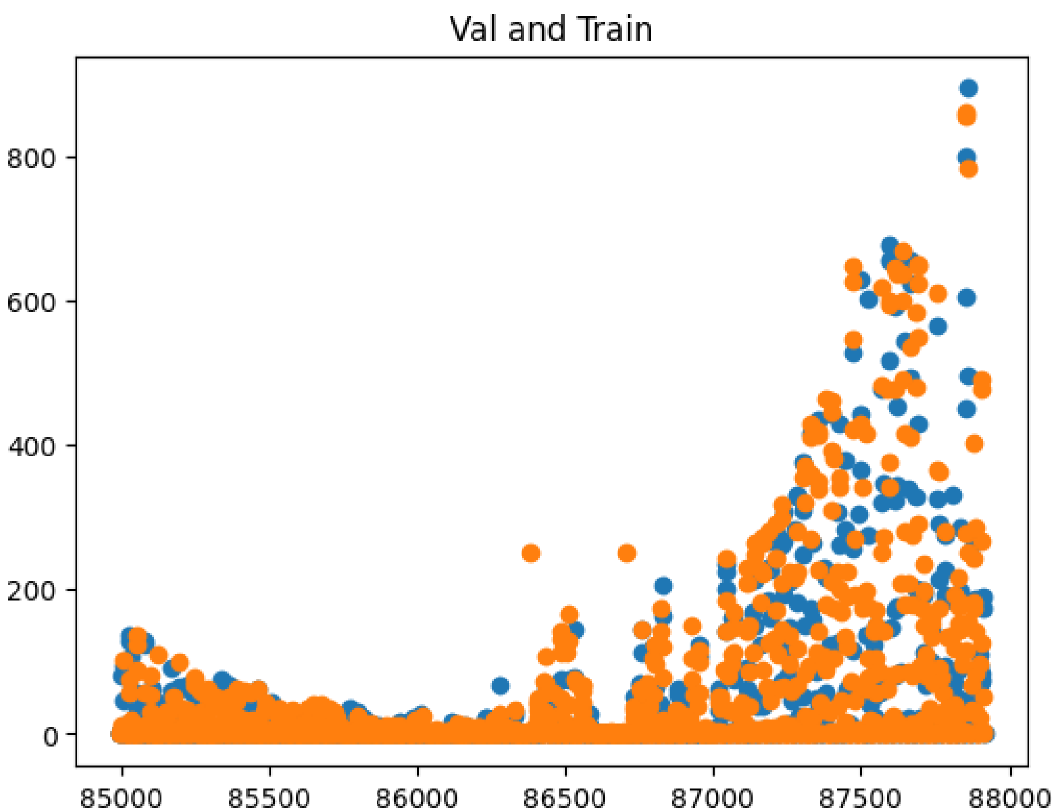
```
Training model for location C…
```

```
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   184.82 GB / 315.93 GB (58.5%)
Train Data Rows:    24073
Train Data Columns: 32
Tuning Data Rows:    1481
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, -0.0, 80.87539, 169.67845)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    129927.71 MB
        Train Data (Original)  Memory Usage: 7.82 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
```

```
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 5.95 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.13s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 599.87s of the
```

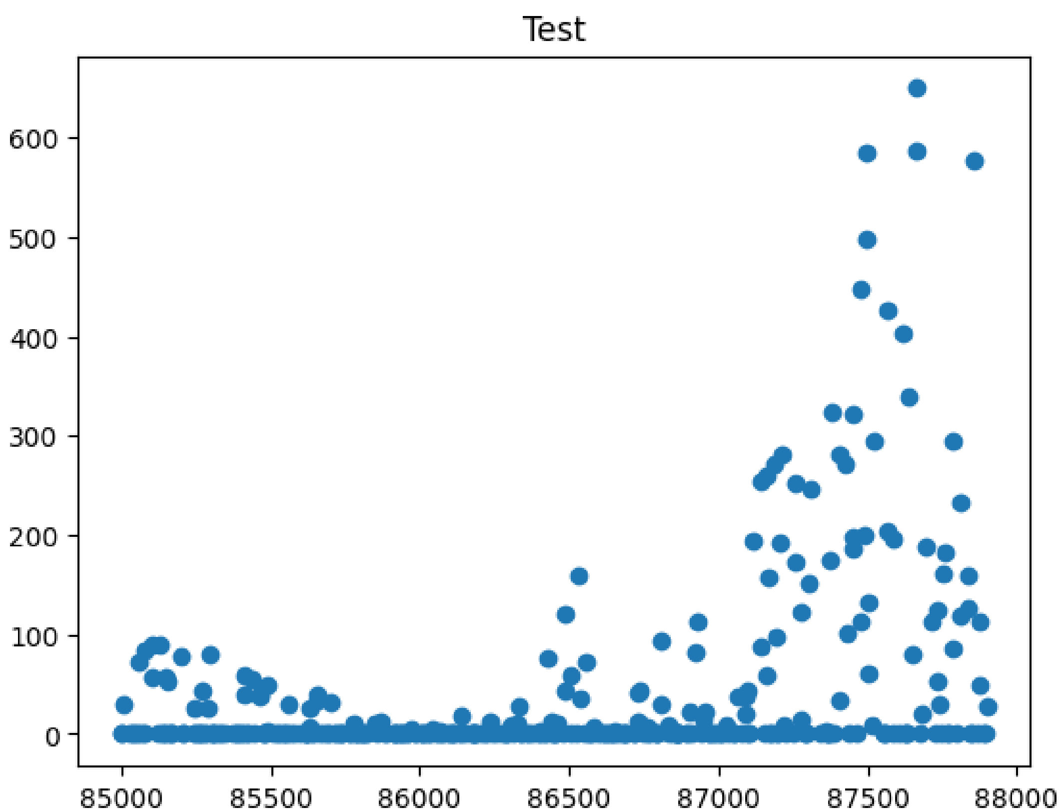
```
599.86s of remaining time.
	-19.8149	 = Validation score   (-mean_absolute_error)
	0.03s	 = Training   runtime
	0.23s	 = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 599.54s of the
599.54s of remaining time.
	-20.1923	 = Validation score   (-mean_absolute_error)
	0.02s	 = Training   runtime
	0.24s	 = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 599.22s of the
599.22s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-12.0011	 = Validation score   (-mean_absolute_error)
	26.66s	 = Training   runtime
	11.62s	 = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 567.83s of the
567.83s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-12.9066	 = Validation score   (-mean_absolute_error)
	28.1s	 = Training   runtime
	11.21s	 = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 535.18s of
the 535.18s of remaining time.
	-16.9074	 = Validation score   (-mean_absolute_error)
	4.64s	 = Training   runtime
	0.74s	 = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 529.18s of the
529.18s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-13.5128	 = Validation score   (-mean_absolute_error)
	187.92s	 = Training   runtime
	0.08s	 = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 339.97s of the
339.97s of remaining time.
	-15.9653	 = Validation score   (-mean_absolute_error)
	1.02s	 = Training   runtime
	0.76s	 = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 337.52s of
the 337.52s of remaining time.
	Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
	-14.908	 = Validation score   (-mean_absolute_error)
	30.49s	 = Training   runtime
	0.39s	 = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 305.51s of the
```

```
305.51s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.6424         = Validation score   (-mean_absolute_error)
        42.09s   = Training   runtime
        2.87s    = Validation runtime
Fitting model: NeuralNetTorch_BAG_L1 … Training model for up to 259.98s of the
259.98s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.2934         = Validation score   (-mean_absolute_error)
        95.4s    = Training   runtime
        0.29s    = Validation runtime
Fitting model: LightGBMLarge_BAG_L1 … Training model for up to 163.05s of the
163.05s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.6543         = Validation score   (-mean_absolute_error)
        88.53s   = Training   runtime
        14.43s   = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 360.0s of the
64.86s of remaining time.
        -11.7779         = Validation score   (-mean_absolute_error)
        0.49s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 535.65s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_117_C/")
Evaluation: mean_absolute_error on test data: -12.584003824628166
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -12.584003824628166,
    "root_mean_squared_error": -31.139472127970958,
    "mean_squared_error": -969.6667244086802,
    "r2": 0.9017109076526874,
    "pearsonr": 0.9525374084963925,
    "median_absolute_error": -0.610007107257843
}

Evaluation on test data:
-12.584003824628166
```



	model	score_test	score_val	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order
0	LightGBMXT_BAG_L1	-12.443105	-12.001052	0.916938	11.622920	26.659214	0.916938	11.622920	26.659214	1	True	3
1	WeightedEnsemble_L2	-12.584004	-11.777877	3.983517	26.574187	211.107298	0.003877	0.000586	0.487236	2	True	12
2	LightGBM_BAG_L1	-13.894913	-12.906645	0.917551	11.206847	28.100699	0.917551	11.206847	28.100699	1	True	4
3	LightGBMLarge_BAG_L1	-14.032733	-12.654305	2.897759	14.425822	88.529264	2.897759	14.425822	88.529264	1	True	11
4	CatBoost_BAG_L1	-14.108424	-13.512758	0.070206	0.075076	187.920821	0.070206	0.075076	187.920821	1	True	6
5	XGBoost_BAG_L1	-14.512669	-13.642395	0.691973	2.873267	42.086031	0.691973	2.873267	42.086031	1	True	9

```
6   NeuralNetFastAI_BAG_L1  -14.527012 -14.908031        0.165590        0.385511
30.492981                 0.165590                0.385511           30.492981
1        True           8
7    NeuralNetTorch_BAG_L1  -15.476211 -14.293353        0.154163        0.291770
95.399875                 0.154163                0.291770           95.399875
1        True          10
8     ExtraTreesMSE_BAG_L1  -15.854892 -15.965258        0.329145        0.756213
1.017239                 0.329145                0.756213           1.017239
1        True           7
9   RandomForestMSE_BAG_L1  -16.879251 -16.907427        0.301368        0.739903
4.643765                 0.301368                0.739903           4.643765
1        True           5
10   KNeighborsUnif_BAG_L1  -20.049167 -19.814903        0.010780        0.233090
0.031708                 0.010780                0.233090           0.031708
1        True           1
11   KNeighborsDist_BAG_L1  -20.130193 -20.192291        0.012307        0.239446
0.023639                 0.012307                0.239446           0.023639
1        True           2
```



[22]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

5 Submit

[23]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

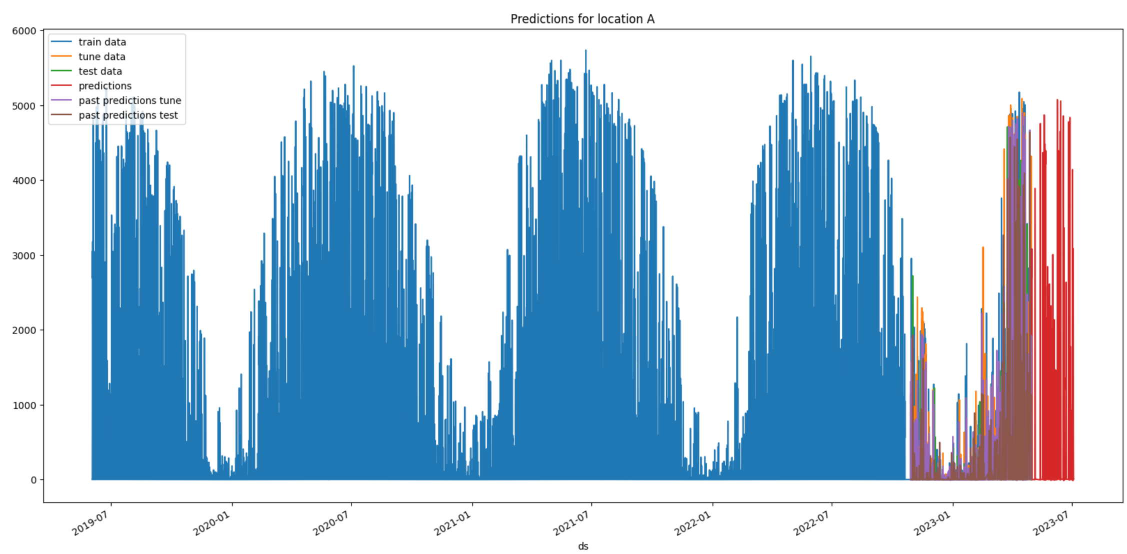
```
Loaded data from: X_test_raw.csv | Columns = 33 / 33 | Rows = 4608 -> 4608
```

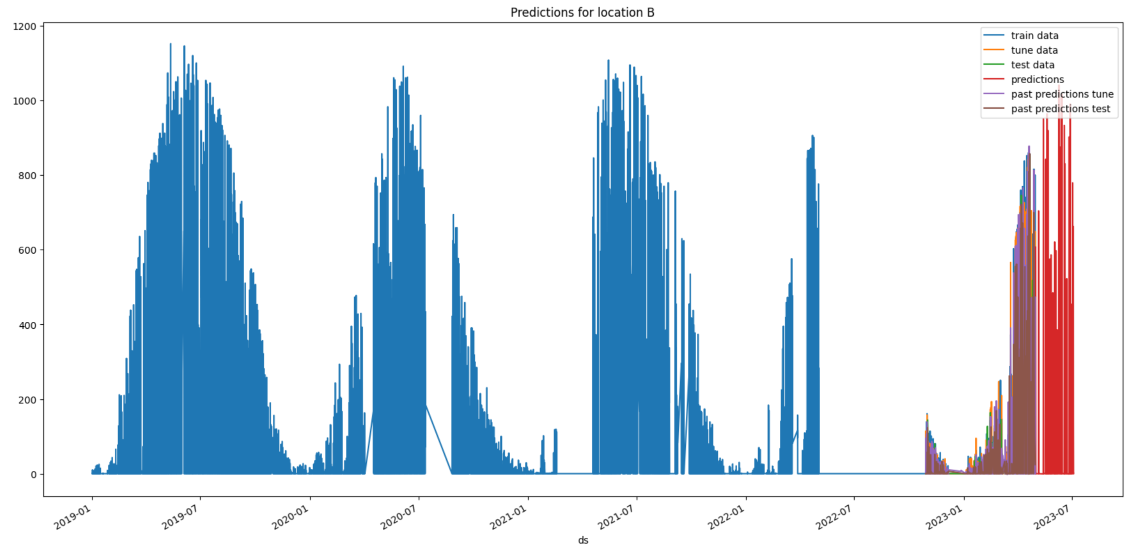
[24]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

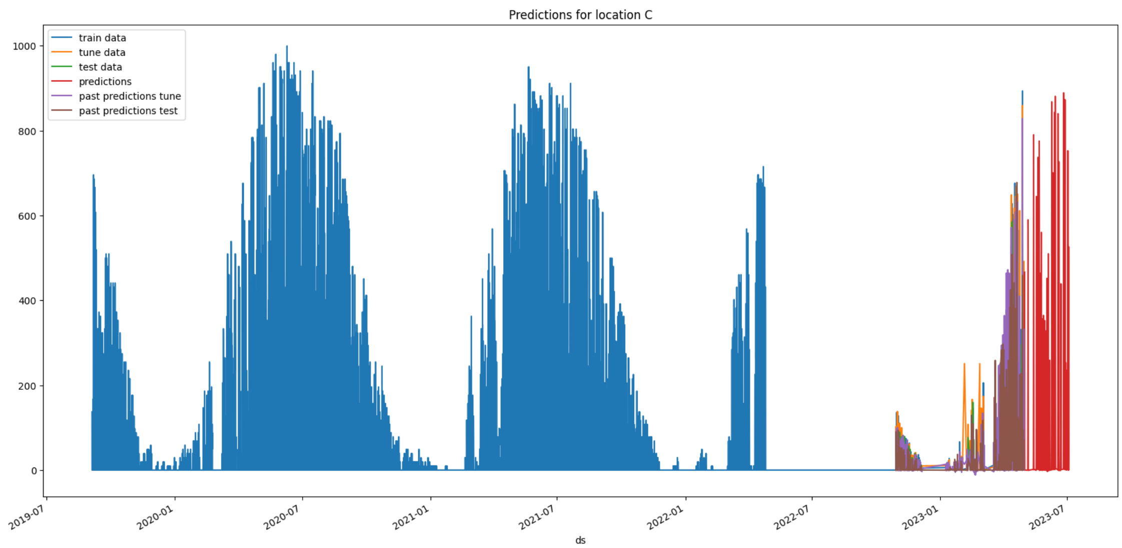
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

[25]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(test_data[test_data["location"] == loc])
```

[26]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction',␣
↪ax=ax, label="past predictions test")


    # title
    ax.set_title(f"Predictions for location {loc}")
```







[27]:
```python
temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())
```

```python
    # Instead of clipping, shift all prediction values up by the largest negative␣
    ↪number.
    # This way, the smallest prediction will be 0.
    elif shift_predictions:
        for pred in temp_predictions:
            # print smallest prediction
            print("Smallest prediction:", pred["prediction"].min())
            pred["prediction"] = pred["prediction"] - pred["prediction"].min()
            print("Smallest prediction after clipping:", pred["prediction"].min())

    elif shift_predictions_by_average_of_negatives_then_clip:
        for pred in temp_predictions:
            # print smallest prediction
            print("Smallest prediction:", pred["prediction"].min())
            mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
            # if not nan
            if mean_negative == mean_negative:
                pred["prediction"] = pred["prediction"] - mean_negative

            pred.loc[pred["prediction"] < 0, "prediction"] = 0
            print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -15.189806
Smallest prediction after clipping: 0.0
Smallest prediction: -0.9926228
Smallest prediction after clipping: 0.0
Smallest prediction: -3.365578
Smallest prediction after clipping: 0.0
```

[27]:

	id	prediction
0	0	0.264840
1	1	0.075831
2	2	0.000000
3	3	22.811451
4	4	314.528473
..	…	…
715	2155	70.236168
716	2156	42.680779
717	2157	9.114732

```
718  2158    1.687364
719  2159    0.859757

[2160 rows x 2 columns]
```

[28]:
```
# Save the submission DataFrame to submissions folder, create new name based on last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"), index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_117.csv
jall1a
```

[29]:
```
# feature importance
# print starting calculating feature importance for location A with big text font
print("\033[1m" + "Calculating feature importance for location A..." + "\033[0m")
print(predictors[0].feature_importance(feature_stage="original", data=test_data[test_data["location"] == "A"], time_limit=60*10))
print("\033[1m" + "Calculating feature importance for location B..." + "\033[0m")
print(predictors[1].feature_importance(feature_stage="original", data=test_data[test_data["location"] == "B"], time_limit=60*10))
print("\033[1m" + "Calculating feature importance for location C..." + "\033[0m")
print(predictors[2].feature_importance(feature_stage="original", data=test_data[test_data["location"] == "C"], time_limit=60*10))
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using 361
rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location A…

```
        1299.44s        = Expected runtime (129.94s per shuffle set)
        139.63s = Actual runtime (Completed 10 of 10 shuffle sets)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using 361
rows with 10 shuffle sets… Time limit: 600s…

                                  importance     stddev       p_value   n  \
direct_rad:W                    1.041988e+02   6.668117   1.427482e-12  10
```

clear_sky_rad:W	7.438919e+01	11.457458	3.601711e-09	10
diffuse_rad:W	7.424264e+01	10.157891	1.263296e-09	10
clear_sky_energy_1h:J	2.928669e+01	9.663301	2.545385e-06	10
sun_elevation:d	2.578224e+01	5.787485	9.717179e-08	10
sun_azimuth:d	2.554878e+01	10.899777	2.025165e-05	10
direct_rad_1h:J	1.711467e+01	4.546942	4.126003e-07	10
effective_cloud_cover:p	1.634965e+01	4.831327	1.014802e-06	10
diffuse_rad_1h:J	1.405516e+01	4.553976	2.189442e-06	10
total_cloud_cover:p	1.282021e+01	4.676614	5.794828e-06	10
snow_water:kgm2	8.816801e+00	5.724266	4.413318e-04	10
relative_humidity_1000hPa:p	7.933399e+00	2.622618	2.585456e-06	10
fresh_snow_6h:cm	5.975481e+00	5.166557	2.628384e-03	10
precip_type_5min:idx	5.466500e+00	5.376921	5.287238e-03	10
ceiling_height_agl:m	5.163184e+00	2.420696	4.206554e-05	10
wind_speed_10m:ms	4.633015e+00	2.376237	8.279019e-05	10
cloud_base_agl:m	4.314251e+00	1.878707	2.377681e-05	10
visibility:m	4.223608e+00	2.627592	3.300455e-04	10
msl_pressure:hPa	4.153987e+00	2.305118	1.474062e-04	10
is_in_shadow:idx	3.769222e+00	1.838298	5.677995e-05	10
is_day:idx	2.950654e+00	1.170744	1.140488e-05	10
sfc_pressure:hPa	2.570660e+00	1.397123	1.267697e-04	10
pressure_100m:hPa	2.529240e+00	1.297129	8.274370e-05	10
pressure_50m:hPa	2.426877e+00	1.599053	4.872830e-04	10
snow_depth:cm	1.879774e+00	1.919162	6.388218e-03	10
t_1000hPa:K	1.496394e+00	2.696151	5.656604e-02	10
super_cooled_liquid_water:kgm2	7.139707e-01	1.847637	1.263771e-01	10
fresh_snow_1h:cm	5.932012e-01	3.353543	2.947785e-01	10
is_estimated	3.381449e-07	0.000000	5.000000e-01	10
fresh_snow_3h:cm	-3.592280e-01	2.956925	6.451162e-01	10

	p99_high	p99_low
direct_rad:W	1.110515e+02	9.734601e+01
clear_sky_rad:W	8.616389e+01	6.261450e+01
diffuse_rad:W	8.468179e+01	6.380350e+01
clear_sky_energy_1h:J	3.921756e+01	1.935583e+01
sun_elevation:d	3.172997e+01	1.983451e+01
sun_azimuth:d	3.675035e+01	1.434721e+01
direct_rad_1h:J	2.178751e+01	1.244183e+01
effective_cloud_cover:p	2.131475e+01	1.138455e+01
diffuse_rad_1h:J	1.873523e+01	9.375090e+00
total_cloud_cover:p	1.762631e+01	8.014106e+00
snow_water:kgm2	1.469956e+01	2.934040e+00
relative_humidity_1000hPa:p	1.062863e+01	5.238166e+00
fresh_snow_6h:cm	1.128509e+01	6.658716e-01
precip_type_5min:idx	1.099230e+01	-5.929925e-02
ceiling_height_agl:m	7.650905e+00	2.675463e+00
wind_speed_10m:ms	7.075045e+00	2.190984e+00
cloud_base_agl:m	6.244975e+00	2.383526e+00

```
visibility:m                    6.923953e+00  1.523263e+00
msl_pressure:hPa                6.522930e+00  1.785044e+00
is_in_shadow:idx                5.658419e+00  1.880025e+00
is_day:idx                      4.153814e+00  1.747495e+00
sfc_pressure:hPa                4.006467e+00  1.134853e+00
pressure_100m:hPa               3.862285e+00  1.196196e+00
pressure_50m:hPa                4.070206e+00  7.835490e-01
snow_depth:cm                   3.852075e+00 -9.252632e-02
t_1000hPa:K                     4.267197e+00 -1.274409e+00
super_cooled_liquid_water:kgm2  2.612766e+00 -1.184824e+00
fresh_snow_1h:cm                4.039598e+00 -2.853196e+00
is_estimated                    3.381449e-07  3.381449e-07
fresh_snow_3h:cm                2.679569e+00 -3.398025e+00
```

Calculating feature importance for location B…

```
        1581.37s        = Expected runtime (158.14s per shuffle set)
        167.84s = Actual runtime (Completed 10 of 10 shuffle sets)
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using 360
rows with 10 shuffle sets… Time limit: 600s…

                                  importance    stddev       p_value   n  \
clear_sky_rad:W                 3.042688e+01  2.295836  6.250438e-12  10
direct_rad:W                    2.321155e+01  1.740872  5.921519e-12  10
diffuse_rad:W                   2.098557e+01  2.537880  4.222386e-10  10
sun_elevation:d                 1.688510e+01  1.599528  4.785686e-11  10
clear_sky_energy_1h:J           1.590629e+01  2.471794  3.896064e-09  10
sun_azimuth:d                   7.962087e+00  1.035239  8.037701e-10  10
direct_rad_1h:J                 7.427158e+00  1.002753  1.122711e-09  10
diffuse_rad_1h:J                6.991924e+00  1.736271  2.317543e-07  10
effective_cloud_cover:p         3.399326e+00  0.474949  1.521179e-09  10
relative_humidity_1000hPa:p     2.469861e+00  0.658992  4.278848e-07  10
is_in_shadow:idx                2.413554e+00  0.562939  1.352469e-07  10
total_cloud_cover:p             2.310786e+00  0.599296  3.359810e-07  10
t_1000hPa:K                     2.187227e+00  0.551104  2.625067e-07  10
snow_water:kgm2                 2.103100e+00  0.750219  4.830639e-06  10
sfc_pressure:hPa                1.693297e+00  0.611724  5.355206e-06  10
snow_depth:cm                   1.333067e+00  0.932913  7.247724e-04  10
msl_pressure:hPa                1.306378e+00  0.433752  2.680452e-06  10
fresh_snow_6h:cm                1.240190e+00  0.757592  2.909597e-04  10
wind_speed_10m:ms               1.037221e+00  0.466554  3.058723e-05  10
pressure_50m:hPa                9.534433e-01  0.326826  3.485856e-06  10
visibility:m                    8.514308e-01  0.319188  7.229994e-06  10
cloud_base_agl:m                6.686370e-01  0.459323  6.422157e-04  10
super_cooled_liquid_water:kgm2  6.629682e-01  0.596669  3.290340e-03  10
pressure_100m:hPa               6.525581e-01  0.562887  2.593294e-03  10
precip_type_5min:idx            4.688106e-01  0.648699  2.406935e-02  10
fresh_snow_1h:cm                4.269509e-01  0.437282  6.490196e-03  10
```

is_day:idx	4.160070e-01	0.229232	1.401034e-04	10
fresh_snow_3h:cm	3.771036e-01	0.310332	1.975155e-03	10
ceiling_height_agl:m	3.247718e-01	0.311629	4.647090e-03	10
is_estimated	-1.056703e-08	0.000000	5.000000e-01	10

	p99_high	p99_low
clear_sky_rad:W	3.278629e+01	2.806748e+01
direct_rad:W	2.500062e+01	2.142247e+01
diffuse_rad:W	2.359372e+01	1.837742e+01
sun_elevation:d	1.852891e+01	1.524128e+01
clear_sky_energy_1h:J	1.844652e+01	1.336606e+01
sun_azimuth:d	9.025990e+00	6.898185e+00
direct_rad_1h:J	8.457676e+00	6.396641e+00
diffuse_rad_1h:J	8.776270e+00	5.207579e+00
effective_cloud_cover:p	3.887425e+00	2.911226e+00
relative_humidity_1000hPa:p	3.147099e+00	1.792623e+00
is_in_shadow:idx	2.992081e+00	1.835028e+00
total_cloud_cover:p	2.926676e+00	1.694896e+00
t_1000hPa:K	2.753590e+00	1.620864e+00
snow_water:kgm2	2.874091e+00	1.332109e+00
sfc_pressure:hPa	2.321959e+00	1.064635e+00
snow_depth:cm	2.291811e+00	3.743242e-01
msl_pressure:hPa	1.752140e+00	8.606170e-01
fresh_snow_6h:cm	2.018759e+00	4.616212e-01
wind_speed_10m:ms	1.516693e+00	5.577494e-01
pressure_50m:hPa	1.289318e+00	6.175682e-01
visibility:m	1.179457e+00	5.234046e-01
cloud_base_agl:m	1.140678e+00	1.965957e-01
super_cooled_liquid_water:kgm2	1.276157e+00	4.977890e-02
pressure_100m:hPa	1.231031e+00	7.408560e-02
precip_type_5min:idx	1.135471e+00	-1.978493e-01
fresh_snow_1h:cm	8.763405e-01	-2.243880e-02
is_day:idx	6.515863e-01	1.804277e-01
fresh_snow_3h:cm	6.960282e-01	5.817894e-02
ceiling_height_agl:m	6.450289e-01	4.514796e-03
is_estimated	-1.056703e-08	-1.056703e-08

Calculating feature importance for location C…

```
        1203.85s        = Expected runtime (120.38s per shuffle set)
        142.28s = Actual runtime (Completed 10 of 10 shuffle sets)
```

	importance	stddev	p_value	n \
clear_sky_rad:W	12.848164	1.051923	1.298497e-11	10
clear_sky_energy_1h:J	8.660572	0.552848	1.395958e-12	10
sun_elevation:d	5.571272	0.557426	7.798270e-11	10
t_1000hPa:K	4.594118	1.203343	3.656218e-07	10
direct_rad:W	3.645789	0.366433	8.120582e-11	10
direct_rad_1h:J	3.197773	0.425498	9.867402e-10	10
sun_azimuth:d	2.691780	0.659642	2.068744e-07	10

diffuse_rad_1h:J	1.895191	0.337909	1.306253e-08	10
diffuse_rad:W	1.180097	0.211176	1.348649e-08	10
relative_humidity_1000hPa:p	0.849099	0.528803	3.324647e-04	10
is_day:idx	0.812707	0.076154	4.341827e-11	10
cloud_base_agl:m	0.649183	0.362763	1.550030e-04	10
total_cloud_cover:p	0.614289	0.337070	1.358972e-04	10
visibility:m	0.599073	0.269099	3.026299e-05	10
effective_cloud_cover:p	0.425750	0.331107	1.407878e-03	10
snow_depth:cm	0.376274	0.184028	5.799294e-05	10
snow_water:kgm2	0.357737	0.234883	4.759453e-04	10
precip_type_5min:idx	0.322209	0.415479	1.830651e-02	10
is_in_shadow:idx	0.303562	0.075978	2.479256e-07	10
ceiling_height_agl:m	0.190540	0.261281	2.326621e-02	10
wind_speed_10m:ms	0.153661	0.328418	8.655861e-02	10
msl_pressure:hPa	0.093277	0.247016	1.314743e-01	10
sfc_pressure:hPa	0.074595	0.140131	6.329799e-02	10
pressure_50m:hPa	0.071263	0.212223	1.579789e-01	10
fresh_snow_6h:cm	0.047521	0.264184	2.916934e-01	10
is_estimated	0.000000	0.000000	5.000000e-01	10
pressure_100m:hPa	-0.014777	0.224994	5.799530e-01	10
fresh_snow_3h:cm	-0.036237	0.078786	9.101101e-01	10
fresh_snow_1h:cm	-0.051935	0.081331	9.628988e-01	10
super_cooled_liquid_water:kgm2	-0.060717	0.104353	9.505398e-01	10

	p99_high	p99_low
clear_sky_rad:W	13.929213	11.767115
clear_sky_energy_1h:J	9.228727	8.092416
sun_elevation:d	6.144132	4.998412
t_1000hPa:K	5.830780	3.357456
direct_rad:W	4.022368	3.269210
direct_rad_1h:J	3.635052	2.760493
sun_azimuth:d	3.369687	2.013874
diffuse_rad_1h:J	2.242456	1.547927
diffuse_rad:W	1.397121	0.963074
relative_humidity_1000hPa:p	1.392544	0.305654
is_day:idx	0.890969	0.734444
cloud_base_agl:m	1.021990	0.276376
total_cloud_cover:p	0.960692	0.267886
visibility:m	0.875623	0.322523
effective_cloud_cover:p	0.766025	0.085475
snow_depth:cm	0.565398	0.187151
snow_water:kgm2	0.599124	0.116350
precip_type_5min:idx	0.749191	-0.104773
is_in_shadow:idx	0.381643	0.225480
ceiling_height_agl:m	0.459055	-0.077975
wind_speed_10m:ms	0.491172	-0.183851
msl_pressure:hPa	0.347132	-0.160577
sfc_pressure:hPa	0.218605	-0.069416

```
pressure_50m:hPa                  0.289362  -0.146837
fresh_snow_6h:cm                  0.319020  -0.223977
is_estimated                      0.000000   0.000000
pressure_100m:hPa                 0.216447  -0.246001
fresh_snow_3h:cm                  0.044730  -0.117205
fresh_snow_1h:cm                  0.031648  -0.135518
super_cooled_liquid_water:kgm2    0.046525  -0.167959
```

[30]:
```
# save this notebook to submissions folder
import subprocess
import os
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),␣
 ↪"autogluon_each_location.ipynb"])
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
 ↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
 ↪ipynb"])
```

```
[NbConvertApp] Converting notebook autogluon_each_location.ipynb to pdf
/opt/conda/lib/python3.10/site-packages/nbconvert/utils/pandoc.py:51:
RuntimeWarning: You are using an unsupported version of pandoc (2.9.2.1).
Your version must be at least (2.14.2) but less than (4.0.0).
Refer to https://pandoc.org/installing.html.
Continuing with doubts…
  check_pandoc_version()
[NbConvertApp] Support files will be in notebook_pdfs/submission_117_files/
[NbConvertApp] Making directory
./notebook_pdfs/submission_117_files/notebook_pdfs
[NbConvertApp] Writing 182756 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no
citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 1921932 bytes to notebook_pdfs/submission_117.pdf
```

[30]:
```
CompletedProcess(args=['jupyter', 'nbconvert', '--to', 'pdf', '--output',
'notebook_pdfs/submission_117.pdf', 'autogluon_each_location.ipynb'],
returncode=0)
```

[31]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
 ↪stderr=subprocess.STDOUT)
```

```
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
 ↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."

# execute_git_command(git_repo_path, ['config', 'user.email',␣
 ↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
 ↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
 ↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
 ↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```

[ ]:

```

```